C++11 code 2.10.12: Function `projectionOntoGradients` for Sub-problem (2-10.h)
→ GitLab

```
2  template <typename FUNCTOR>
3  Eigen::VectorXd projectOntoGradients(const lf::assemble::DofHandler &dofh,
4                                       FUNCTOR f) {
5    const lf::assemble::size_type N_dofs = dofh.NumDofs();
6    Eigen::VectorXd sol_vec;
7
8    // I. Build the (full) Galerkin matrix for the linear system.
9    lf::assemble::COOMatrix<double> A(N_dofs, N_dofs);
10   ProjectionOntoGradients::ElementMatrixProvider my_mat_provider;
11   // co-dimension 0 because we locally assemble on cells
12   lf::assemble::AssembleMatrixLocally(0, dofh, dofh, my_mat_provider, A);
13
14   // II. Build the (full) right hand side vector
15   Eigen::VectorXd phi(N_dofs);
16   phi.setZero();
17   ProjectionOntoGradients::GradProjRhsProvider my_vec_provider(f);
18   // co-dimension 0 because we locally assemble on cells
19   lf::assemble::AssembleVectorLocally(0, dofh, my_vec_provider, phi);
20
21   // III. Enforce homogeneous dirichlet boundary conditions
22   // We do this by selecting the DOFs on the boundary and setting the
23   // values to zero.  Note, that we could also use this to set the
       boundary
24   // to any other value (if bc were not homogeneous)
25
26   // To select the right DOFs we need a selector:
27   // * for all DOFs on the boundary return true and the value on the
       boundary
28   // * for all other DOFs return false (and whatever as value)
29   const double boundary_val = 0;
30   auto bd_flags{lf::mesh::utils::flagEntitiesOnBoundary(dofh.Mesh(), 2)};
31   auto my_selector = [&dofh, &bd_flags, &boundary_val](unsigned int dof_idx) {
32     if (bd_flags(dofh.Entity(dof_idx))) {
33       return std::make_pair(true, boundary_val);
34     } // interior node:  the value we return here does not matter
35     return std::make_pair(false, 42.0);
36   };
37   // Since we know the values on the boundary we know the solution on
       these
38   // DOFs and we can write the Galerkin LSE in block format and solve
       only
39   // for the unknown coefficients.  This modification is done by the
       following
40   // function FixFlaggedSolutionComponents().  We use the selector we
       have
41   // defined above.
42   // See [Lecture → Section 2.7.6] for explanations.
43   lf::assemble::FixFlaggedSolutionComponents<double>(my_selector, A, phi);
44
45   // IV. Solve the LSE using an Eigen solver
46   // Convert from triplet to CRS format
47   const Eigen::SparseMatrix<double> A_crs = A.makeSparse();
48   Eigen::SparseLU<Eigen::SparseMatrix<double>> solver;
49   solver.compute(A_crs);
50   sol_vec = solver.solve(phi);
51   return sol_vec;
52 }
```